April 1996

UILU-ENG-96-2210
CRHC-96-06

Center for Reliable and High Performance Computing

Fault Dictionary Compaction Using Structural and Tree-Based Techniques

Vamsi Boppana

Coordinated Science Laboratory
College of Engineering
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS None
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) UILU-ENG-94-2210 CRHC-96-06	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Coordinated Science Lab University of Illinois 6b. OFFICE SYMBOL *(If applicable)* N/A	7a. NAME OF MONITORING ORGANIZATION 1) Semiconductor Research Corporation 2) Office of Naval Research
6c. ADDRESS *(City, State, and ZIP Code)* 1308 W. Main St. Urbana, IL 61801	7b. ADDRESS *(City, State, and ZIP Code)* Research Triangle Park, NC 27709 800 N. Quincy St., Arlington, VA 22217
8a. NAME OF FUNDING/SPONSORING ORGANIZATION 7a 8b. OFFICE SYMBOL *(If applicable)*	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS *(City, State, and ZIP Code)* 7b	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE *(Include Security Classification)*
Fault Dictionary Compaction Using Structural and Tree-Based Techniques

12. PERSONAL AUTHOR(S)
Vamsi Boppana

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT *(Year, Month, Day)*	15. PAGE COUNT
Technical	FROM ______ TO ______	April 1996	59

16. SUPPLEMENTARY NOTATION

17. COSATI CODES		
FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*
fault dictionary, dictionary compaction, fault diagnosis, testing, diagnostic experiment trees, tree encoding

19. ABSTRACT *(Continue on reverse if necessary and identify by block number)*

For repeated diagnosis of different copies of the same circuit, the fault dictionary is typically better than dynamic diagnosis, because it requires substantially less time. However, there are problems associated with the dictionary approach. One of the most significant problems with the use of dictionaries for diagnosis is the size problem. The sizes of the dictionaries that have to be stored are large and impractical for even reasonably large circuits. Hence, there is a necessity for compact fault dictionaries that are feasible to be constructed for large circuits in reasonable time.

In this work, fault dictionary compaction has been addressed from the point of view of two relatively (but not entirely) orthogonal tasks. The first of these tasks has been identified as identifying diagnostically useful information. Previous techniques attempting to solve this problem had several limitations, thus limiting their applicability to large practical circuits. Efficient techniques were proposed to identify diagnostically useful information, thus resulting in dictionaries with satisfactory resolution and are feasible to be generated on large practical circuits.

(OVER)

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT [✓ UNCLASSIFIED/UNLIMITED] ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE *(Include Area Code)* / 22c. OFFICE SYMBOL

Abstract Cont.

The second task is one of representing identified information efficiently. This task is especially significant in applications where the first task cannot be performed. This is true in many faulty chips where the fault modeling process is not accurate, i.e., to say that the presence of unmodeled faults could have caused the observed errors. A novel approach to storing all the information in the full fault dictionary based on the use of unlabeled tree encoding has been proposed as an alternative to reducing the size of storage considerably. The proposed storage structure uniquely captures the indistinguishability class information at various stages of the diagnosis process and, hence, is shown to exhibit better behavior than currently known techniques for the purpose of matching the faulty symptoms with the stored data.

FAULT DICTIONARY COMPACTION USING STRUCTURAL AND TREE-BASED TECHNIQUES

BY

VAMSI BOPPANA

B.Tech., Indian Institute of Technology, Kharagpur, 1993

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1995

Urbana, Illinois

ACKNOWLEDGEMENTS

I wish to express a deep sense of gratitude towards my advisor Professor W. Kent Fuchs for his invaluable guidance, encouragement and support throughout the entire course of this work. I also wish to thank all of the members of the Center for Reliable and High Performance Computing for the excellent work environment. Last, but not least, sincere thanks are due to my parents for providing me with the opportunity to pursue my dream, and to my sister for constantly encouraging me.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

The aim of testing is to determine whether a circuit is faulty. This is necessary for ensuring the correct operation of digital systems. Once a circuit is found to be faulty, fault location or *diagnosis* is performed on the faulty system to locate the physical failure. Diagnosis may be intended for identification and replacement of a faulty subcircuit or may be performed with a view to improving the manufacturing process.

1.1 Diagnosis

1.1.1 Problem

The *input* to a diagnostic routine is a defective integrated circuit chip and it *aims* to obtain a subset of faults that can explain all the errors observed while testing the chip. The motivation behind this process is to minimize the number of probes that is required to identify the faults in the defective chip.

1.1.2 Classification

Diagnosis has been considered in previous work [2], [3]. Diagnostic techniques can be broadly classified into two groups. The first group uses methods that depend on pre-computed information using fault dictionaries [9], [10], [13]. The second group uses methods that dynamically diagnose the faulty behavior of the circuit, while the test set is applied [4].

In the first approach, a dictionary is first computed, such that for every *modeled fault*, the response of the circuit in the presence of the fault is given in sufficient detail to allow the fault location process to determine whether or not that fault explains the faulty behavior of the circuit. In the second approach (dynamic diagnosis), the response of the circuit to a test is analyzed through fault simulation to determine the fault that matches the observed test results.

The two approaches can also be combined to create a pre-computed dictionary to eliminate some faults followed by dynamic fault diagnosis which is used to increase the resolution of the diagnostic process [11]. Diagnosis is usually performed under a specific fault model. Unmodeled faults cannot be located with 100% confidence. When the response cannot be explained by modeled faults, the fault best matching the response is selected. Another approach involves observing the internal lines of the circuit [12]. This has the advantage of being independent of the fault model, but is may not be cost-effective.

1.2 Motivation for This Work

For repeated diagnosis of different copies of the same circuit, the fault dictionary is typically better than dynamic diagnosis, because it requires substantially less time. However, there are problems associated with the dictionary approach. One of the most significant problems with the use of dictionaries for diagnosis is the size problem.The sizes of the dictionaries that have to be stored are large and impractical for even reasonably large circuits. Hence, there is a necessity for compact fault dictionaries that are feasible to be constructed for large circuits in reasonable time.

There exist several known file compaction techniques such as *gzip* or *compress*, but they are not applicable to this specific problem because of two reasons. First, the compaction offered by these techniques is not enough to bring the sizes of the dictionaries to be manageable and second, they require the entire dictionary to be created before compaction and uncompacted before use. Image and video compression techniques provide another possible alternative, but lossy techniques are generally not useful. This creates the need for compaction techniques specifically targeting fault dictionaries.

Previous fault dictionary compaction techniques address the size problem, but have the following problems:

1. It is not feasible to use any previous compaction technique yielding dictionaries of reasonable diagnostic capability for large practical circuits, as they are severely limited by memory and run-time considerations.

2. Even in applications when it is possible to apply a compaction technique, the diagnostic performance of the resulting dictionary is unclear because the information discarded by the compaction technique could be vital in the presence of un-modeled faults.

1.3 Work Done in the Thesis

In this thesis, solution techniques are proposed for the above identified problems of dictionary compaction. Chapters 3 and 4 present solutions to the first problem. The first technique identifies output pin information that does not have to be stored with respect to modeled faults in the circuit. This technique, unlike previous techniques, uses *structural analysis* to identify the information to be discarded. This makes it faster than previous compaction techniques. Next presented is a technique to identify and eliminate output sequences from dictionaries resulting in little or no loss of diagnostic resolution with respect to modeled faults. This provides a reduction in both the space required to store the dictionaries and the time required to compute the dictionaries as compared with previous techniques. For each of the schemes, experimental results are presented on benchmark circuits to demonstrate their efficiency. Chapter 5 presents a solution to the second problem. The proposed storage scheme uses *unlabeled tree* encoding to provide all the information in a conventional dictionary. This technique provides a storage alternative in applications where elimination of any diagnostic information may be undesirable.

2. DEFINITIONS AND PREVIOUS WORK

2.1 Definitions

Fault dictionaries are introduc⁀d in this chapter. A full fault dictionary is first explained followed by the introduction of the diagnostic experiment tree, which represents the entire diagnostic experiment. Several previous compaction techniques are then reviewed.

2.1.1 Full fault dictionary

The full fault dictionary is a record of the errors a circuit's modeled faults are expected to cause on each output after each test vector. It is generated through fault simulation without fault dropping; defects are located by comparing the errors observed on a faulty chip to those recorded in the dictionary, to find the faults causing errors most like those observed.

Full fault dictionaries consider each output separately for each vector. For a circuit with V vectors, O outputs and F faults, a full fault dictionary is conventionally stored as a matrix containing VO columns and F rows. Each matrix element indicates whether some fault is detected, not

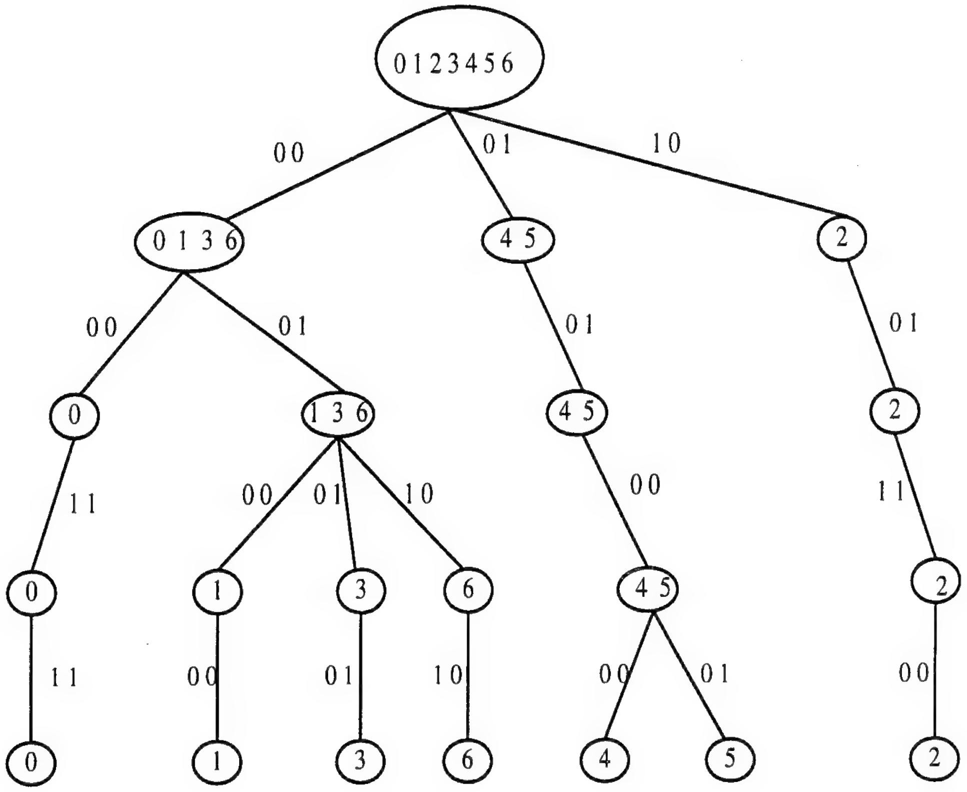
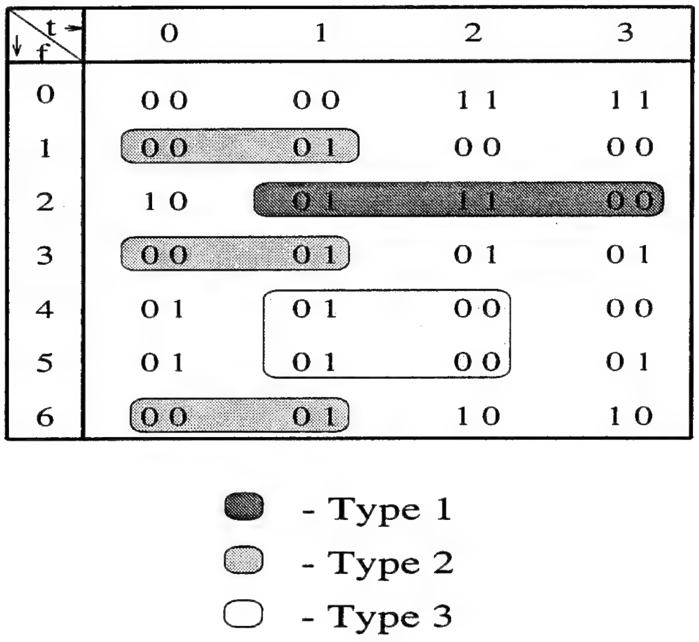


Figure 2.1: Diagnostic experiment tree.

detected or potentially detected. The potentially detected entry is required only for sequential circuits. As an example, consider the full fault dictionary for the diagnostic experiment of Figure 2.1, shown in Figure 2.2. The circuit has seven faults and two primary outputs. The number of test vectors used for this example is four.

It would be ideal to store the full fault dictionary and use it in the diagnostic process because compaction techniques may lose some diagnostic resolution, but their sizes tend to be prohibitively large for even moderately large circuits. As an example, the full fault dictionary for the circuit $s35932$ from the ISCAS89 benchmark circuits requires about 9532 Mb of storage. Hence there is a need for smaller dictionaries that can perform reasonably well diagnostically.



f ↓ \ t →	0	1	2	3
0	0 0	0 0	1 1	1 1
1	0 0	0 1	0 0	0 0
2	1 0	0 1	1 1	0 0
3	0 0	0 1	0 1	0 1
4	0 1	0 1	0 0	0 0
5	0 1	0 1	0 0	0 1
6	0 0	0 1	1 0	1 0

● - Type 1
▨ - Type 2
○ - Type 3

Figure 2.2: Full fault dictionary showing types of redundancies.

2.1.2 Representation of the diagnostic experiment

Definition 1 (Diagnostic Experiment Tree $T(V, E)$**)** *The diagnostic experiment tree consists of a set of vertices $V(T)$ and a set of directed edges $E(T)$, i.e., each edge e is of the form (u, v), $u, v \in V(T)$ with the direction of the edge being* from u to v. *Each vertex $v \in V(T)$ of the tree is associated with a set of faults $F(v)$, which is a subset of the list of all modeled faults F, and each edge $e \in E(T)$ is associated with a list of outputs $O(e)$, which is a subset of all the primary outputs of the circuit.*

The root of the tree defines the start of the diagnostic experiment.

Definition 2 (Root of the Diagnostic Experiment Tree $r(T)$**)** *The unique node* $r(T) \in V(T)$ *such that there is no edge of the form* $(u, r(T))$, $\forall u \in V(T)$ *is the root of the tree* T.

The fault list associated with such a node is the entire list of modeled faults of the circuit, i.e., the set F. The application of various test vectors in the experiment can be represented by using the *levels* of the tree.

Definition 3 (Level of a node $L(v)$**)** *For each node* $v \in V(T)$ *of the tree, the level* $L(v)$ *is defined as the length of the path between* $r(T)$ *and* v.

Definition 4 (Parent of a node v $P(v)$**)** *For each node in a tree T except the root* $r(T)$, *there is a unique node* $P(V)$ *such that the edge* $(P(v), v) \in E(T)$.

With the above definitions, it is trivial to represent the diagnostic experiment by the diagnostic experiment tree. Let $T(V, E)$ be the tree representing the diagnostic experiment. The start of the experiment is represented by the root $r(T)$ with the entire fault list F being associated with it. A test vector is applied as part of the experiment at each level of the tree. At a given level i of the circuit, for each node v, the node $P(v)$ at level $i - 1$ represents the node containing all the faults in $F(v)$. The output response associated with the edge $(P(v), v)$ is the unique output response produced by each of the faults in $F(v)$. The don't care value is treated as a special symbol of the alphabet.

Example : In Figure 2.1, the diagnostic experiment tree is shown representing information from the full fault dictionary shown in Figure 2.2, represented in the conventional matrix format.

2.2 Previous Fault Dictionary Compaction Techniques

Pass/Fail [14] : This type of fault dictionary records the faults detected, potentially detected and not detected for each vector. It does not record detections separately by output. It is created by a single full fault simulation and is much smaller than a full fault dictionary. But, as might be expected, this dictionary loses some diagnostic capability when compared with the full fault dictionary.

Compact [14] : One method of enhancing the diagnostic capability of the pass/fail dictionary is to add output information. Such an approach is used in the creation of the compact fault dictionary. This dictionary is created by adding columns for specific outputs and vectors to the pass/fail dictionary. To select the additional columns, the faults not completely distinguished by the vector dictionary are re-simulated, counting for each vector and output the number of new pairs distinguished. A separate column is added to the dictionary for the vector and output pair distinguishing between the most new fault pairs, repeating for the still undistinguished faults, until no new pairs can be distinguished.

The compact algorithm is computation intensive, requiring multiple simulations of all vectors against some faults, plus a full fault simulation to produce the vector dictionary and another to produce the final dictionary after extra columns are added. The dictionary produced is known to be considerably compressed, with no loss of resolution [15]. Since the algorithm chooses columns by selecting the output that distinguished between most fault pairs, the algorithm is essentially a greedy algorithm and does not necessarily result in the optimal number of columns.

Sequential [15] : In this technique, a pass/fail dictionary is enhanced by a single full fault simulation. An entry is added to the dictionary for any vector and output that distinguishes between any pair of faults not previously distinguished. This is computationally cheaper that the Compact algorithm. Each vector and output are considered once, sequentially, and two entries may be included where one later entry would distinguish between the same faults. There is no loss of resolution.

List Splitting [15] : This dictionary is created by using efficient list splitting. However, it is not accurate for sequential circuits; hence, the diagnostic resolution suffers.

Drop on K [15] : While creating this dictionary, the fault simulator drops each fault after its Kth detection and creates an otherwise standard dictionary, including possible detections until each fault's Kth definite detection. This technique assumes that K detections distinguish between most fault pairs and that some faults cause errors for many vectors, filling dictionaries with unneeded data. Simulation costs here are less than for a full fault dictionary.

First Failing Pattern [15] : This is a special case of the Drop on K dictionary for K equal to 1.

Detection Frequencies [15] : A full fault simulation is performed, and for each fault f, the number of vectors definitely (d_f) and potentially producing errors (p_f) are counted. Each fault can cause errors numbering between d_f and $d_f + p_f$. The list of faults that causes each possible number of errors forms an indistinguishability class for this dictionary. The resolution of this dictionary is poor in comparision with other schemes.

In this work, small dictionaries are proposed to overcome the drawbacks of the previous methods. Structural analysis of the circuit is exploited to identify redundant output information.

Next, redundant output sequences are identified and eliminated to generate dictionaries of small sizes even while retaining high diagnostic resolution with respect to modeled faults.

It must be noted at this point that none of the above compaction algorithms provide for any compaction when it is not possible to discard any information from the full fault dictionary. A solution to this problem is proposed in Chapter 5, where full fault dictionaries are stored using unlabeled tree-based tree encoding rather than the conventional matrix-based approach.

3. OUTPUT PIN REMOVAL

3.1 Introduction

It has already been observed that fault dictionaries are prohibitively large even for reasonably large circuits. A brief review of the expended research effort in generating fault dictionaries of reasonable size without compromising their diagnostic resolution greatly has also been presented. One of the main problems with such techniques was the considerable computational effort necessary to generate these dictionaries. In this chapter, a method based on structural analysis for identifying output pin information not requiring storage in combinational circuits is presented. It is also proposed that the use of this information can result in a decrease in the dictionary size without excessive computation. This is demonstrated by a simple dictionary generation scheme. Experimental results on the ISCAS 85 benchmark circuits are presented in support of the above claims.

3.2 Identification of Removable Output Pins

3.2.1 Removable output pins

In this section, a method for structurally identifying removable output pin information is presented. The essential idea that governs this process is that each fault in the circuit affects only a subset of the primary outputs; hence, there is no need to store information for the rest of the primary outputs for that fault in any fault dictionary.

Let us consider a circuit C. Let the total number of lines in the circuit be equal to L and let them be numbered from $1, \ldots, L$. Let the set of inputs of C be denoted by I and the set of outputs be denoted by O. *Structural dependency* for a pair of lines $(i, j), 1 \leq i \leq L, 1 \leq j \leq L$ in the circuit is first defined.

Definition 5 (Structural Dependency $SD(i,j)$ **)** *Line* $j, 1 \leq j \leq L$ *is* **structurally dependent** *on line* $i, 1 \leq i \leq L$*, if there can be a path for a signal* 0 *or a* 1 *to propagate from the line* i *to* j*. We define in such a case* $SD(i,j) = 1$*, otherwise,* $SD(i,j) = 0$.

Therefore, we see that the structural dependency gives information as to whether a value at a line can potentially influence the value at another line in the circuit. We can extend the concept of *structural dependency* and define *cones of influence* which can be used to identify the output redundancy in the circuit. There exists a *cone of influence* for each line $i, 1 \leq i \leq L$, in the circuit.

Definition 6 (Cone of Influence $COI(i)$ **)**

$$COI(i) = \{j | 1 \leq j \leq L \text{ and } SD(i,j) = 1\}$$

The *cone of influence* of a line is the set of all those lines that are structurally dependent on that line. We can now go one step further and say that a *cone of influence* exists for each fault in the circuit, because each line in the circuit can have two faults as per our fault model: *s-a-0 and s-a-1*.

We note that each line $i, 1 \leq i \leq L$, has a subset of outputs included in its *cone of influence*. Let us define this subset of outputs called the *Necessary Outputs* for the node i.

Definition 7 (Necessary Outputs $NO(i)$)

$$NO(i) = \{j | j \in O \; and \; SD(i,j) = 1\}$$

We see that the set $NO(i)$ of a line $i, 1 \leq i \leq L$, gives the set of outputs of the circuit that are potentially affected by a fault in the circuit.

The following theorem proves that for any line $i, 1 \leq i \leq L$, no other output apart from the ones in the set $NO(i)$ will be affected under the influence of a fault in line i. Let the set of output lines of C be denoted by $O = (O_1, \ldots, O_n)$ and the values of the output lines of C under the influence of no fault be $(O_1 = o_1, \ldots, O_n = o_n)$. Let the values of the output lines of C under the influence of a fault f be denoted by $(O_1 = of_1, \ldots, O_n = of_n)$.

Theorem 1 *Let the line i have a fault f in it. Then, the values at the output lines of C are given by*

$$\begin{aligned} O_j &= of_j, j \in O, j \in NO(i) \\ &= o_j \; , j \in O, j \notin NO(i) \end{aligned}$$

Proof 1 *There are two cases to prove in this theorem. The first case corresponds to output lines in the set $NO(i)$ and the second case corresponds to the lines not in the set $NO(i)$.*

Case 1

This case is trivially true from the definition of the set $(of_1, \ldots, of_n)$.

Case 2

Let us assume if possible, that a line $O_j, j \notin NO(i)$, takes the value $of_j \neq o_j$. This implies that the line j is being structurally affected by a fault in the line i. This implies that $SD(i, j) = 1$, which implies that $j \in NO(i)$. Hence, we have reached a contradiction. Therefore, we must have $of_j = o_j$, which proves Case 2.

From the above theorem, we can conclude for any line i in the circuit that the values at the output lines that do not lie in the *cone of influence* of line i, do not change under the influence of a fault at line i.

In Figure 3.1, we have $NO(Lx) = (O1, O2)$, whereas $NO(Ly) = (O2)$. Hence, we see that we need not store values of both $O1, O2$ for faults at all lines. In conclusion, we have seen in this subsection, that no fault dictionary has to store more output information than what is identified as necessary by this scheme. We shall next see how to construct the sets $NO(i)$ for each line i in the circuit. A simple fault dictionary that uses this information to eliminate structural output redundancy is presented later.

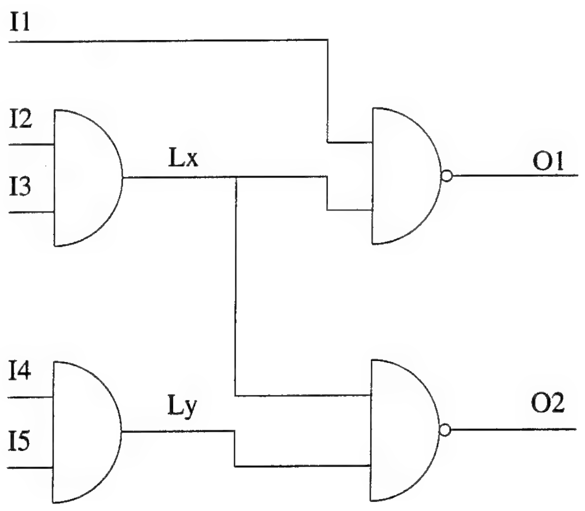


Figure 3.1: Example of necessary outputs.

3.2.2 Computing NO information

The algorithm used for computing the necessary output information is presented here. To efficiently implement this algorithm, we immediately note that there it is not necessary to consider the lines that are inputs to logic gates, because of the simple fact that the set of circuit outputs that have to be observed for each input of the gate will be the same as that required for the output of that gate. Hence, only lines that are outputs of gates are considered. We call them *nodes*. Once, the NO information for all the nodes is computed, then, information for all the lines can be updated in a single pass. The following definitions are required before an algorithm can be presented.

Definition 8 (Successors $S(i)$ **)** *This is defined for each* node i. *The outputs of all gates to which the node i is fanning out to are included in the set $S(i)$.*

It should be noted that the set $S(i) = \phi$ for each output line.

Definition 9 (Predecessors $P(i)$) *This is also defined for each* node i. *The outputs of all gates that are fanning in as input lines of the gate of which i is the output line are included in the set $P(i)$.*

It should be noted that the set $S(i) = \phi$ for each input line (also included as a node).

Definition 10 (Level $L(i)$) *The level of a node is defined as the maximum number of gates needed to be traversed for reaching the node.*

Another improvement in the algorithm is obtained by the topological sorting procedure, which orders the circuit by grouping sets of nodes with the same *level*. As can be easily seen, each input node will have a level of 0.

ComputeNO

```
1. Set $NO(i)$ for each node to be $\phi$;
2. Sort the circuit topologically and arrange the circuit in the topologically leveled order;
3. Enter each output node into a queue and mark it solved;
4. If queue is empty, then go to 10;
5. i = removequeue();
6. If all elements in $S(i)$ are not solved, enterqueue($i$); go to 4;
7. Add to $NO(i)$ the set $NO(j)$ for each $j \in S(i)$ without adding repetitions;
8. enterqueue($j$),for each $j \in P(i)$;
```

9. *Mark i solved; go to 4;*

10. *Assign $NO(i)$ for each line that is an input to a gate by copying the $NO(j)$ set where j is the output node of the gate to which i is an input;*

11. *Done.*

The above algorithm gives the set $NO(i)$ for each line in the circuit. As we can readily see, this is a polynomial time algorithm, because each node can enter the queue a maximum number of times p, which is upper bounded by the maximum fanout of a node in the circuit. Hence, if there are n nodes in the circuit, then the complexity of Steps 3 to 9 is $O(n * p)$. Steps 2 and 10 also take polynomial time, each taking $O(L)$ time. Each line of the circuit has to be examined a constant number of times. A simple fault dictionary construction scheme that eliminates redundant output information is next presented.

3.3 Computing Small Fault Dictionaries

In the previous section, a scheme that can identify structural output redundancy in the circuit was presented. This information can be efficiently used in the construction of a small fault dictionary called the *Structurally Output Irredundant Fault Dictionary, SOIFD*. The organization of this dictionary is in the form of a two-dimensional matrix. The rows correspond to the test vectors. Each column corresponds to a unique (fault,output) pair. The outputs to be stored for a particular fault are decided by the algorithm **ComputeNO** described earlier. The faults are elements of the set of all equivalent faults and cover the whole set.

The following scheme is used for generating the fault dictionary.

GenerateSOIFD

1. *Generate diagnostic test vectors for the given circuit description using any standard package. The scheme does not change with different test vectors; i.e., given any set of test vectors, we can compute our dictionary using that set of vectors.*
2. *Compute the set of equivalent faults using any standard package.*
3. *Determine the NO information for the set of equivalent faults using* **ComputeNO**.
4. *Perform full fault simulation with the given set of test vectors and the given set of equivalent faults and generate the information for each fault as indicated by the corresponding NO.*

This simple scheme has been implemented and experimental results are compared with other schemes in the next section.

3.4 Experimental Results

Experiments were performed on the ISCAS 85 benchmark circuits to demonstrate the effectiveness of the scheme. The sizes of the fault dictionaries obtained using SOIFD were compared with those obtained using compact [COMP.] [14] and also with the full fault dictionary FFD. COMP. was chosen as the basis for comparision with a small fault dictionary, because it reportedly gives very small dictionary sizes when compared with other schemes [15] and has the same diagnostic capability as the full fault dictionary.

The benchmark circuits were first levelized. The NO information is generated for each line in the circuit. SOIFD, as explained earlier, is applicable to any given test set. In order to make a fair comparison between SOIFD and COMP. the same test set is used as in COMP. The SOIFD is generated by fault simulating with the given test set and the set of equivalent faults and by obtaining output information for only outputs in the set $NO(i)$ for each fault f and line i. The results on the sizes of dictionaries are summarized in Table 3.1.

In Table 3.1, N_T is the number of test vectors in the test set. N_F is the number of faults in the reduced fault list F. N_C is the number of columns added by COMP. to enhance the resolution of a vector dictionary to that of a full fault dictionary. Thus, $(N_F * (N_T + N_C))$ gives the size of the COMP. dictionary. The column SO gives the sum total of all outputs that need to be observed for each test vector in the test set, i.e., SO = $\sum_{\forall f_i \in F} NO(i)$ where f is a fault in line i. Hence, the size of the SOIFD is given by (N_T*SO). The size of the full fault dictionary (FFD) is given by $N_F * N_T * N_O$, where N_O is the number of outputs in the circuit. The sizes of the dictionaries computed are in terms of number of *bits*.

From Table 3.1 it can be observed that the full fault dictionary contains a considerable amount of redundant information. For example, about 99% of the information in c2670 is redundant. This is due to the fact that the circuit has a large number of outputs and lines and not all of the outputs need to be observed for most of the faults. It has also been seen that the SOIFD gives smaller sizes than COMP. for five out of the ten cases. This shows the potential that SOIFD has in obtaining small fault dictionaries.

Table 3.1: Experimental Results on ISCAS 85 circuits.

Cir.	N_T	N_F	N_C	N_O	SO	SOIFD	COMP.	SOIFD/ COMP.	SOIFD/ FFD
1. c432	44	524	53	7	2523	111012	52574	2.184	0.687
2. c499	60	758	35	32	11552	693120	72010	9.652	0.476
3. c880	30	942	92	26	3179	95370	114924	0.828	0.129
4. c1355	95	1574	25	32	24412	2319140	188880	12.278	0.484
5. c1908	142	1877	106	25	19912	2110672	465496	6.074	0.316
6. c2670	167	2748	153	139	9917	664439	879360	0.755	0.0104
7. c3540	111	3422	332	22	24678	2740257	1515946	1.807	0.327
8. c5315	56	5354	461	123	34582	1936592	2768018	0.699	0.0525
9. c6288	16	7744	216	32	92084	1473344	1796608	0.820	0.371
10. c7552	87	7550	525	108	40007	3480609	4620600	0.754	0.049

It is clear from the above data that the information required to be stored in both SOIFD and COMP. is far less than that required for FFD for many large circuits. We can also see that the sizes of both SOIFD and COMP. are similar. It should be noted again here that SOIFD requires far less computational overhead than COMP. This implies that the SOIFD technique can obtain similar sizes of fault dictionaries with far less computation.

3.5 Conclusions

In this chapter, a scheme for the identification of structural output redundancy has been presented. A simple scheme for generating a fault dictionary by eliminating the identified redundancy has also been presented. Experimental results on the ISCAS 85 benchmark circuits show that the sizes of the dictionaries obtained are comparable to the COMP. dictionaries. However, the computational effort required is considerably lower, because the COMP. fault dictionary requires multiple fault simulations in contrast to the single one required by our scheme. The problem with this technique is that it is only applicable to combinational circuits. Experiments performed for extending this idea to sequential circuits were not encouraging. Hence, in the next chapter, we propose a solution based on the use of the diagnostic experiment tree that solves this problem efficiently for sequential circuits.

4. OUTPUT SEQUENCE REMOVAL

4.1 Introduction

In the previous chapter, we discussed a technique for generating dictionaries of small size, but was only applicable to combinational circuits. In this chapter, techniques for eliminating entire sequences of outputs and for efficiently storing the remaining output sequences are presented. Experimental results on the ISCAS 85 and ISCAS 89 benchmark circuits show that the sizes of dictionaries proposed are substantially smaller than for the full fault dictionary, while the dictionaries retain most or all of the diagnostic capability of the full fault dictionary (we are concerned with modeled faults in this chapter also).

4.2 Output Sequence Removal

There are three types of output sequences that we can potentially remove from the dictionary.

4.2.1 Type 1

If a node i at level p of the diagnostic tree has only one fault in the list of faults, then we need not continue the diagnostic experiment for combinational circuits as we have already found the required fault. Hence, it is unnecessary to store the output sequence produced by the fault in the list at node i for all vectors $p, \ldots, (l-1)$. It has been noted in our experiments that such sequences can also be dropped for sequential circuits, even if the node i is not fully distinguished, without any significant loss in the resolution. In this implementation for sequential circuits, partially specified output responses are handled by treating X as a separate alphabet and having pointers whenever necessary from other nodes in the tree.

4.2.2 Type 2

For a node i at level p the full fault dictionary stores the output responses produced by each of the faults in the list of faults at i for all vectors $0, \ldots, p-1$. Each of these output responses is the same; otherwise, the faults would not be members of the same list at node i. Hence, the same output sequence is replicated in the full fault dictionary. It is shown in Section 4.3 that this information can be stored without replication.

4.2.3 Type 3

If the application of the test sequence $p_{n_1}, \ldots, p_{n_k}$ to a node i at level p results in a single node j, then all the faults in the lists at both nodes i and j are the same for combinational circuits.

Hence, this test sequence is not useful for distinguishing between any pair of faults in the list at node i. This implies that during diagnosis, if we are at node i, then the observation of output responses produced by each of the tests in this sequence is unnecessary. This can be utilized to reduce the size of the dictionary as well as to reduce the number of test responses needed to be observed and will result in a reduction in the number of tests to be applied to diagnosis if the circuit is combinational. It has also been observed during out experiments that such sequences can also be dropped for sequential circuits, without any significant loss of resolution.

4.2.4 Example

The following example illustrates the three types of removable output sequences identified above. Figure 2.1 shows the diagnostic experiment tree for a fault dictionary with seven faults, four vectors and two output bits. The full fault dictionary with all the types of output sequences is shown in Figure 2.2.

Type 1: In Figure 2.1, fault 2 has been distinguished from all other faults after the application of test 0. Hence , the output responses for fault 2 for test vectors 1,2 and 3 can be eliminated.

Type 2: In Figure 2.1, the output sequence 00 01 can be stored only once for all the faults 1, 3 and 6.

Type 3: In Figure 2.1, tests 1 and 2 do not have any effect on the node (4,5) and, hence, do not provide useful output information for distinguishing between faults 4 and 5.

4.3 Small Dictionaries

4.3.1 Common subsequence CSS(i)

The common subsequence of a node i at level p in the diagnostic experiment tree is defined as the sequence of output responses due to vectors $0 \ldots (p-1)$, say S, iff *there is no subsequence S_1 of S that is of type 1.*

In Figure 2.1, the CSS of the node with the list of faults (1 3 6) is 00 01. Every Common Subsequence inherits another common subsequence called the Inherited Common Subsequence defined below.

4.3.2 Inherited CSS ICSS(i)

Let the node whose CSS number is i be n. Then, the inherited common subsequence of the common subsequence i is defined as the sequence of output responses due to vectors $0 \ldots j$, where $j+1$ is the least level where a node m can be found with a fault list exactly identical to the list at node n.

In Figure 2.1, the CSS 01 01 00 of the node with the fault list (4 5) inherits the ICSS 01, because j as defined above equals 0. Let us now introduce two dictionaries $DC1$ and $DC2$, with $DC1$ eliminating sequences of types 1 and 2 and $DC2$ eliminating sequences of types 1, 2 and 3. $DC2$ is smaller than $DC1$ in most but not all cases.

4.3.3 Dictionary $DC1$

This dictionary is comprised of a vector and a table. The vector is called the $FVector$ and the table is the $CSSTable$. Each of these can be represented in the compact bit packed representation below.

FVector

This vector has F entries, where F is the total number of collapsed single stuck-at faults in the circuit. The entry $FVector[i]$ in this vector gives the common subsequence associated with the fault i when it is at a leaf of the diagnostic experiment tree. For example, in $DC1$ constructed for the diagnostic experiment of Figure 2.1 shown in Figure 4.1, we have CSS number 7 stored for fault 3.

CSSTable

Each entry in this table is indexed by the common subsequence number. Each entry in this table has three fields. The first field in the entry $CSSTable[i]$ has the inherited subsequence number ICCS(i). The second field has the number of vectors whose output response sequence when concatenated with the output sequence obtained from the the inherited common subsequence gives the common subsequence i. The third field is a bit stream comprised of the actual output responses that when concatenated with the output sequence obtained from the inherited common subsequence gives the common subsequence i. This is a variable length field in the dictionary, but the storage can still be in a bit packed manner, because we know exactly how many bits are

FVector

	CSS
0	4
1	6
2	3
3	7
4	9
5	10
6	8

CSSTable

	ICSS	# ops	op responses
1	0	1	00
2	0	3	01 01 00
3	0	1	10
4	1	1	00
5	1	1	01
6	5	1	00
7	5	1	01
8	5	1	10
9	2	1	00
10	2	1	01

Figure 4.1: $DC1$ for the tree in Figure 2.1.

present in this field from the second field in the same entry.

As an example, in Figure 4.1, the entry for CSS 4 has ICSS=1, the number of vectors whose output responses are stored in this entry is 1 and the output response is 00. Hence, to construct the CSS 4, we go to the entry for CSS 1, given by the inherited CSS field, and find that the inherited CSS is 0 (which means that there is no inherited CSS). Hence the output sequence of CSS 1 is 00, which implies that the output sequence of CSS 4 is 00 00.

4.3.4 Dictionary $DC2$

This dictionary is comprised of a vector and two tables. The vector is called the $FVector$ and the two tables are called $DTable$ and the $CSSTable$. Again, each of these can be represented in a compact bit packed representation.

FVector

This vector is *identical* to the $FVector$ constructed in $DC1$.

DTable

The rows of the $DTable$ are indexed by fault numbers and the columns are indexed by test numbers. One bit is maintained for each fault, test vector pair. The entry $DTable[i][j]$ is 0, iff the fault lists that include the fault i before and after the application of the test j are the same. It is through the use of this table that we can decide at diagnosis time, whether the application of a test vector is essential. In Figure 4.2, the entries, $DTable[4][1] = 0$ and $DTable[5][1] = 0$, FVector

	CSS
0	4
1	6
2	3
3	7
4	9
5	10
6	8

CSSTable

	ICSS	opseq
1	0	00
2	0	01
3	0	10
4	1	00
5	1	01
6	5	00
7	5	01
8	5	10
9	2	00
10	2	01

DTable

	0	1	2	3
0	1	1	0	1
1	1	1	1	0
2	1	0	0	0
3	1	1	1	0
4	1	0	0	1
5	1	0	0	1
6	1	1	1	0

Figure 4.2: $DC2$ for the tree in Figure 2.1.

imply that the fault list containing 4,5 (i.e., (4 5)) does not change after the application of test vector 1.

CSSTable

There are only two fields for each entry $CSSTable[i]$. The first and third fields of $DC1$ are present, but the second field is absent, because a single output response is stored in the third field instead of the multiple output responses, since information on useful tests is already available from $DTable$. This is how the dictionary $DC2$ eliminates sequences of Type 3.

As an example, the entry for CSS 2 in $DC2$ has just 01 in contrast to the 01 01 00 stored by $DC1$. But, we see that the only two CSSs that inherit CSS 2 are 9 and 10, and from $FVector$ we see that they correspond to faults 4 and 5. Now, from $DTable[4][1] = 0, DTable[5][1] = 0, DTable[4][2] = 0\ and\ DTable[5][2] = 0$, we can conclude that the sequence 01 00 is not needed for the diagnostic experiment as are the tests 1 and 2 for distinguishing between faults in the list (4 5).

4.3.5 Diagnostic capabilities

In both of the dictionaries introduced above, the information eliminated is redundant if the circuit is combinational. For sequential circuits, $DC2$ has been constructed by dropping sequences, even if the node under consideration in the diagnostic tree has not been fully distinguished. It has been noted in our experiments that this does not cause any significant change in the resolution of the dictionary. Hence, $DC1$ and $DC2$ have the same diagnostic capability of the full fault dictionary for combinational circuits, whereas $DC1$ alone retains this capability for sequential circuits. It should be noted that the output responses eliminated may be important for the identification of some non-modeled faults.

4.3.6 Generation of $DC1$ and $DC2$

$DC1$ and $DC2$ have been generated through a traversal of the diagnostic experiment tree. A single full fault simulation without fault dropping is performed to dynamically construct the diagnostic experiment tree. The process of constructing the tree and the dictionaries can be done dynamically, one level at a time. This removes the need for excessive storage when generating the dictionaries.

4.4 Experimental Results

Experiments were performed on the ISCAS 85 and ISCAS 89 benchmark circuits to study the sizes and performances of the dictionaries $DC1$ and $DC2$. A measure for estimating the amount of reduction in the number of test responses that have to be observed during diagnosis for combinational circuits is also presented. This measure also gives an estimate on the average number of tests that have to be applied for diagnosis. The *useful test ratio* is defined as : $useful\ test\ ratio = 1 - (no\ of$ type-3 $seq. - no\ of$ type-1 $seq)/(no\ of\ seq - no\ of$ type-1 $seq)$. For the example of Figure 2.1, we have the *useful test ratio* = $10/12$.

Table 4.1 presents the results for ISCAS 85 circuits whereas the results for ISCAS 89 circuits are

Table 4.1: Experimental results on ISCAS 85 circuits.

Cir.	FF (Kb)	PF/ FF%	DC1/ FF%	DC2/ FF%	UT RAT. %
c432	201	14.3	19.4	23.6	26.2
c499	1334	3.1	9.5	7.7	40.3
c880	1836	3.9	11.9	7.3	20.5
c1355	4432	3.1	27.9	5.1	3.6
c1908	13132	4.0	14.3	4.9	3.8
c2670	85126	0.7	5.3	1.1	10.9
c3540	21739	4.5	9.2	5.5	5.6
c5315	173725	0.8	9.2	1.5	6.2
c6288	11399	3.1	18.9	8.9	20.9
c7552	366930	0.9	11.9	1.3	2.8

given in Table 4.2. In Tables 4.1 and 4.2, the first column (FF) gives the size of the full fault dictionary. The test set used in the creation of the dictionaries was from the HITEC test generator. Experiments were performed only on circuits whose test sets have reasonable diagnostic resolution. The column PF/FF gives the percentage ratio of the size of the Pass/Fail dictionary to the full fault dictionary. The next two columns give the percentage ratios of the sizes of $DC1$ and $DC2$ with respect to the full fault dictionary. The next column in Table 4.1 gives the percentage *useful test ratio*, whereas the next two columns in Table 4.2 give the diagnostic resolutions of the full dictionary and $DC2$. Although not shown, the diagnostic expectations [15] of $DC2$ and the full fault dictionary are very similar except for cases with low diagnostic resolution.

From Tables 4.1 and 4.2, we can make several observations. The sizes of the dictionaries $DC1$ and $DC2$ are significantly smaller than for the full fault dictionary. The size of $DC2$ is almost always smaller than that of $DC1$. The sizes of the dictionaries presented are less than even the

Table 4.2: Experimental results on ISCAS 89 circuits.

Cir.	FF (Mb)	PF/ FF%	DC1/ FF%	DC2/ FF%	Res FF%	Res DC%
s298	0.9	16.7	55.0	9.2	94.522	94.520
s344	0.8	9.1	66.1	6.5	96.771	96.764
s641	4.7	4.2	62.4	2.9	97.764	97.640
s713	4.7	4.4	51.2	2.9	96.275	96.081
s820	31.2	5.2	65.1	2.8	99.571	99.571
s832	31.9	5.2	64.9	2.8	99.388	99.388
s1238	18.1	7.1	59.5	4.1	99.710	99.710
s1423	1.3	20.0	11.4	11.5	59.440	59.140
s5378	405.9	2.0	46.3	1.2	89.993	89.709
s35932	9532.6	0.3	41.4	0.4	98.025	98.025

Pass/Fail dictionary size for many cases. This is significant because the sizes of both the Compact [14] and the Sequential [15] dictionaries are lower bounded by the size of the Pass/Fail dictionary. However, both Compact and Sequential achieve the resolution of the full fault dictionary when compared to the small loss suffered by $DC2$.

4.5 Conclusions

Three kinds of sequences that can be eliminated from fault dictionaries were identified. Two dictionary schemes that eliminate these redundancies were presented along with experimental results on the ISCAS bench mark circuits that show that these dictionaries give a substantial reduction in the size of the dictionary with little or no loss of resolution. It has also been shown that the number of tests whose outputs have to be observed is far less than the full test set size for a large number of diagnosis experiments as indicated by the low *useful test ratios*. These ratios also indicate that the number of tests that have to be applied for diagnosing combinational circuits may be very small.

5. UNLABELED TREE ENCODING

5.1 Introduction

Previous chapters have developed techniques for identifying diagnostically useful information based on modeled faults. However, the process of compaction can remove information that is potentially useful in locating unmodeled failures [8]. The focus of this chapter is on developing a storage structure that can efficiently represent full fault dictionaries without discarding any output information. While developing this technique, care has been taken to ensure that the diagnostic information stored can be easily accessed as opposed to known techniques that store the same information.

In this approach, the labeled tree associated with the diagnostic experiment is represented by storing the label information and the unlabeled tree in disjoint data-structures. The unlabeled tree is represented by a binary string code and the node ordering obtained by the unlabeled tree encoding is used to store the label information efficiently. The unique knowledge of the diagnostic tree that is embedded into the storage structure is fully utilized to aid in the efficient matching

Table 5.1: Matrix representation of a full fault dictionary.

faults	output values	
0	00	11
1	00	11
2	01	01
3	00	10
4	01	00
5	10	11
good	11	00

of the faulty symptoms exhibited by defective circuits. Analysis is presented demonstrating the efficiency of the tree-based representation. Experimental results on the ISCAS 85 and ISCAS 89 circuits present the storage requirements of the proposed structure compared to techniques for storing the full fault dictionary and verify the theoretical predictions for its performance.

5.2 The Storage Structure

The diagnostic experiment tree, introduced earlier, serves as the motivation for the storage structure described in this section. We shall use the same representation of the diagnostic experiment as presented in Chapter 2.

Example : In this chapter, the following example will be used to explain the storage structure. In Figure 5.1, a diagnostic experiment tree is shown representing information from the full fault dictionary shown in Table 5.1, represented in the conventional matrix format.

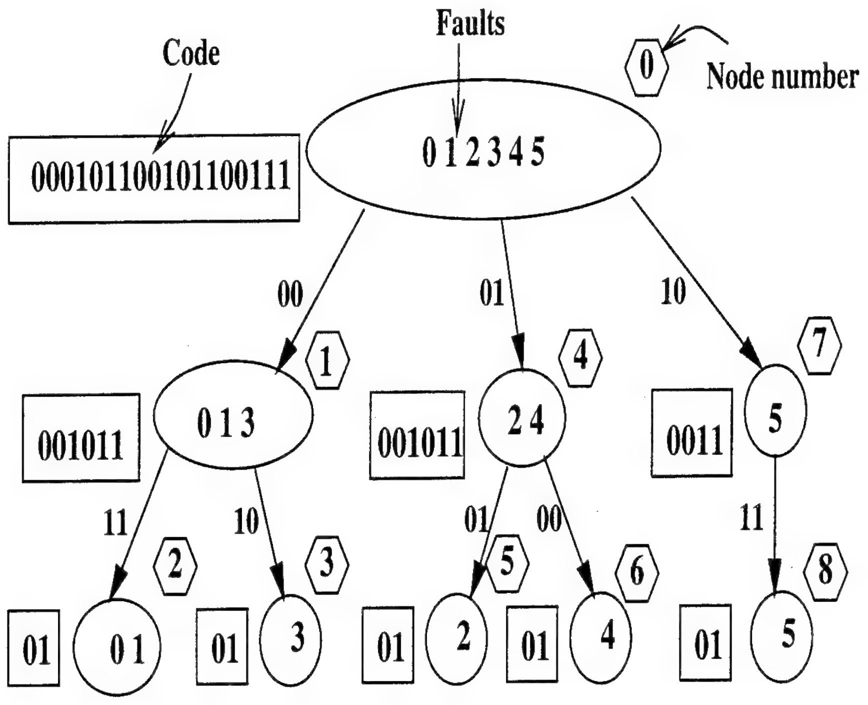


Figure 5.1: Diagnostic experiment tree for the dictionary in Table 5.1.

5.2.1 Dictionary representation

It is clear that the tree $T(V, E)$ represents a labeled tree and can be represented using standard graph-theoretic techniques for storing labeled trees such as Prüfer sequences [6], but the storage required is not necessarily optimal, due to the inherent redundancy in the natural labeling of the tree. Hence, the storage representation used here represents the label information and the underlying unlabeled tree information disjointly.

Let the tree $G(V, E)$ represent the tree corresponding to the diagnostic tree $T(V, E)$, but with its nodes and edges not associated with any information. In $T(V, E)$, each node has a set of faults that is associated with it and each edge has a set of faults associated with it. Let a canonical numbering $Number(v)$ be associated with each node $v \in G(V, E)$. The same numbering is used both in $T(V, E)$ and $G(V, E)$. This numbering is obtained from the scheme used to encode the tree $G(V, E)$. The information in the tree $T(V, E)$ is represented by encoding the tree $G(V, E)$ using a binary code [1].

The tree $G(V, E)$ lacks two pieces of information that are present in $T(V, E)$. First, there is no output information present, and second, there is no information as to the faults present at each internal node of the circuit. The output information is obtained from the tables called the **DERRTABLE**, storing the distinct error responses produced by the circuit and the **INDEXTABLE** storing for each node in the tree. The index leading into the **DERRTABLE** indicates the unique error response associated with that edge of the tree. It has been noted [7] that this technique of storing the output responses as opposed to storing the exact output responses is typically more economical. The second piece of information that has to be maintained in the storage structure in order that the original information is preserved is the set of faults associated with each node of the circuit. It can be shown that with the structure of the unlabeled tree already available (in the form of the binary string), it suffices to store the canonical labeling number of the node in the diagnostic tree at the highest level for each fault. This is exactly the information maintained in the table **FARRAY**.

The following results prove that the information present in the storage structure is indeed complete.

Lemma 1 *There exists a unique sequence of node numbers from any node at the highest level to the root* $r(T)$.

Proof: *This result follows from the fact that there is a unique path from $r(T)$ to any node in the tree.*

Property 1 *The information in the tree $T(V, E)$ can be represented by the given storage scheme.*

Proof: *It suffices to prove that all the output responses associated with each fault in the tree $(T(V, E)$ can be decoded from the present form of storage. From* Lemma 1, *the path from $r(T)$ to any node at the highest level is decipherable; and for each fault, there is a scheme for storing the number of the highest level node containing it. These numbers can then be used to obtain the actual output sequences because they are also stored in the canonical numbering order. Hence the proof.*

5.2.2 Binary code for $G(V, E)$

This section presents the critical unlabeled tree encoding algorithm.

Encoding Algorithm

The encoding is defined inductively over the number of levels in the tree. The encoding of a tree with one vertex and no edges is given by the binary code 01. The encoding of every tree $G(V, E)$ with root $r(G)$ is defined as follows: Let $C_1, C_2, \ldots, C_k$ represent the codes of the subtrees rooted at all the nodes in the tree $G(V, E)$ such that $r(G)$ is their parent. Let these codes be written in some preferred order, to be defined, given by a permutation $(i_1, i_2, \ldots, i_k)$ of the integers $1, 2, \ldots, k$. Then the code of the tree is defined to be $0, C_{i_1}, C_{i_2}, \ldots, C_{i_k}, 1$. A straightforward choice for the preferred order in this application is the order obtained from diagnostic

Table 5.2: INDEXTABLE.

3	3	2	4	1	0	1	3

Table 5.3: DERRTABLE.

0	00
1	01
2	10
3	11

fault simulation.

Example : The binary encoding of each node in the tree in Figure 5.1 is shown adjacent to each node. The **INDEXTABLE**, **DERRTABLE** and the array **FARRAY** are shown in Tables 5.2, 5.3, and 5.4, respectively. To illustrate, consider the retrieval of primary outputs produced by fault 0 at test vector 2. **FARRAY** is used to indicate the distinct error produced by the fault on the last test vector. Then, unlabeled tree encoding is used to obtain the canonical number of the node containing the fault 0 at the required test vector. The canonical number here is 2. This is used to index into **INDEXTABLE**, which, in turn, implies that the distinct error produced is contained in location 3 of the **DERRTABLE**. The error is identified as 11 and knowing the good circuit response to be 00, the outputs produced are identified to be 11.

The following decoding algorithm is presented to prove that the canonical labelling information that has been used in proving that the information stored in this storage structure is complete can indeed be obtained.

Table 5.4: FARRAY.

fault number	node number
0	2
1	2
2	5
3	3
4	6
5	8

A Decoding Algorithm

1. Associate a label with each 0 occurring in the code, by numbering them in order left to right.

2. Scan the code from left to right until the configuration 001 is found. Note the pair of labels associated with the two 0s in this configuration, and then delete the second 0 and the 1. The two labels will define an edge of the tree.

3. If the resulting string has more than two symbols, repeat Step 2, otherwise, the string is just 01, the label associated with this 0 is that of the root $r(T)$ and the algorithm terminates.

The amount of storage required for the storage representation presented above is $2n$ bits, where n is the number of nodes in the tree.

5.3 Performance Analysis of the Storage Structure

The analysis presented in this section establishes the utility of the tree-based storage structure compared to the known techniques to store the full fault dictionary. First, the tree-based technique is shown to perform well with respect to the storage requirements for the full fault dictionary in the matrix format. The analysis of the variation of the size of the list representation [15] versus the matrix representation has been presented in an earlier work [16]. Then, the advantages of the method over the list of faults representation and the matrix representation during faulty symptom matching are presented.

5.3.1 Size performance

For analyzing the space performance of the tree representation, extremal cases are isolated for analysis. It is assumed, for the purpose of analysis, that the output information is represented using the actual primary output responses and not the error responses. Let the number of outputs, faults and test vectors of the circuit be o, f, t, respectively. Also, assume that the circuits in consideration are combinational. This is just to avoid case analysis and the appropriate ratios can be obtained for sequential circuits by multiplying by a factor of 2 to account for the don't care outputs in the appropriate terms.

Upper bound on the ratio

Consider a hypothetical scenario in which the diagnostic experiment tree $T(V, E)$ is just a path. The matrix representation of the full fault dictionary requires oft bits. As opposed to this, the tree-based representation requires no more than $ot + 2t$ bits. Thus, the ratio of their sizes is bounded by $fo/(o+2)$, i.e., $O(f)$. It is clear that over all trees, this ratio forms a bound because the tree-based representation stores more information than in any other case.

Towards a lower bound on the ratio

It is not immediately obvious how to obtain a lower bound as in the upper bound case, but the following scenario is informative. Consider the case in which the *forest* obtained by deleting the root $r(T)$ from $T(V, E)$ is a collection of f paths, with the root of each of these paths having exactly one fault associated with it in $T(V, E)$. This would require a storage of oft bits in the matrix based scheme, whereas $ft(\log(ft)) + 2ft + f(\log(ft))$ with the first term representing the space requirement for the output responses (because the number of distinct output responses is bounded by $\log(ft)$), the second term representing the storage required for the binary encoding and the last term measuring the storage requirement for the array *FARRAY*. This assumes that $\log(ft) < o$. Although it may not be easy to show that this scenario indeed gives a lower bound, it is clear that tree-based representations in such cases give storage of the same order as matrix representations.

The above analysis demonstrates that the storage required by the tree-based representation as opposed to other representations decreases with an increase in the average number of faults associated with each node of the tree. This prediction is verified by performing experiments for computing the size of the tree-based representation on benchmark circuits with a subset of the vector set generated. The performance of the list-based fault dictionary versus the tree-based fault dictionary for the benchmark circuits is also presented.

5.3.2 Matching performance

The storage representations are analyzed here to prove that the tree-based representations requires fewer matching operations than both the list representation and the matrix representations.

Consider any stage of the diagnostic process, i.e., after the application of a number of diagnostic vectors. Let the node corresponding to this stage of the diagnosis be p. Let the number of faults associated with p be equal to f_p. Let the number of children of p be c_p. Then, the number of output comparisions required by the list representation and the matrix representation is equal to f_p and the number of output comparisions required by the tree-based representation is equal to c_p, because the tree-based representation alone has the unique knowledge of the indistinguishability classes at each level of the diagnostic experiment tree. The proof follows because $c_p \leq f_p$ independent of p and the node p in this argument is an arbitrary node in the diagnostic experiment tree.

5.4 Experimental Results

Experiments were performed on the ISCAS 85 and ISCAS 89 benchmark circuits to study the space requirements of the tree-based encoding scheme compared with the matrix and the list schemes. The tree-based encoding scheme has been integrated into the fault simulator and a

Table 5.5: ISCAS 85 circuits studied.

Cir.	f	o	t
c499	758	32	55
c880	942	26	75
c1355	1574	32	88
c1908	1876	25	280
c2670	2595	64	236
c3540	3126	22	316
c5315	5350	123	264
c6288	7744	64	46
c7552	7550	108	450

level-by-level processing of faults is used to generate the new tree-based dictionary. Tables 5.5 and 5.6 represent the circuit characteristics of the ISCAS 85 and ISCAS 89 circuits, respectively. The columns indicate the circuit name, number of equivalent faults, number of primary outputs and the number of test vectors. In Tables 5.7 and 5.8, the sizes of the representations of the full fault dictionary using the tree-based, matrix and list representations are presented for ISCAS 85 and ISCAS 89 circuits, respectively. The columns in these tables indicate the circuit name, total number of nodes in the diagnostic tree, errbits($= \lceil \log(number\ of\ distinct\ errors\ produced) \rceil$), size of the matrix representation, size of the list representation and the size of the tree-based representation. Tables 5.9 and 5.10 represent the results on a few of the circuits with 10 and 20 test vectors, respectively. The columns for the ratios show that the tree-based representation performs significantly better than the matrix representation with fewer test vectors.

Table 5.6: ISCAS 89 circuits studied.

Cir.	f	o	t
s344	342	11	108
s420	455	1	166
s526	555	6	192
s641	467	24	211
s713	581	23	175
s820	850	19	968
s832	870	19	967
s953	1079	23	14
s1238	1355	14	478
s1423	1515	5	88
s5378	4603	49	900
s35932	39094	320	381

5.5 Conclusions

The experimental results for the ISCAS 85 and ISCAS 89 circuits illustrate that full fault dictionaries can be efficiently stored without discarding any information. In the approach presented, a labeled tree was associated with the diagnostic experiment. The labeled tree is represented by disjointly storing the label information and the underlying unlabeled tree. The unlabeled tree is represented by a binary string code, and the node ordering obtained by the unlabeled tree encoding is used to store the label information. Analysis was used to establish the efficiency of the tree-based storage scheme both with respect to the space required and the performance during a matching algorithm.

Table 5.7: Performance of tree-based representation on ISCAS 85 circuits.

Cir.	no. nodes	errbits	matrix	list	tree	tree/matrix
c499	31954	10	1334080	264176	384993	0.2886
c880	49664	9	1836900	247424	539307	0.2936
c1355	63240	10	4432384	650672	747025	0.1685
c1908	364793	10	13132000	3051984	4353787	0.3315
c2670	329169	11	39194880	2759488	4078749	0.1041
c3540	749774	12	21731952	5109232	10133869	0.4663
c5315	1089879	13	173725200	5572720	16687241	0.0961
c6288	239464	11	22798336	4340230	3303826	0.1449
c7552	2523546	13	366930000	14888048	35978193	0.0981

Table 5.8: Performance of tree-based representation on ISCAS 89 circuits.

Cir.	no. nodes	errbits	matrix	list	tree	tree/matrix
s420	9698	2	75530	8000	44824	0.5935
s526	12958	6	639360	403984	99254	0.1552
s641	61205	10	2364888	394928	744137	0.3147
s713	51743	10	2338525	377184	612669	0.2620
s820	535546	10	15633200	2453088	5986269	0.3829
s832	544824	10	15984510	2471584	6122696	0.3830
s953	3747	9	347438	8624	75977	0.2187
s1238	384998	10	9067660	867664	4335675	0.4781
s1423	13270	6	666600	84096	125599	0.1884
s5378	1917705	13	202992300	15187840	27439247	0.1352
s35932	6161291	18	2147483647	280043344	212342918	0.0988

Table 5.9: Performance of tree-based representation with 10 vectors.

Cir.	no. nodes	errbits	matrix	list	tree	tree/matrix
c2670	4652	9	1660800	142016	96120	0.0579
c5315	9909	10	6580500	196320	281118	0.0427
c6288	18443	9	4956160	904288	339922	0.0686
c7552	12271	10	8154000	307360	314375	0.0386
s1238	1083	7	189700	18288	24665	0.1300
s1423	322	5	75750	7312	14947	0.1973
s5378	4623	9	2255470	98640	139932	0.0620
s35932	19534	11	125100800	520288	1919516	0.0153

Table 5.10: Performance of tree-based representation with 20 vectors.

Cir.	no. nodes	errbits	matrix	list	tree	tree/matrix
c2670	13665	9	3321600	288272	204515	0.0616
c5315	31141	11	13161000	409456	613702	0.0466
c6288	71394	10	9912320	1756336	1008889	0.1018
c7552	39142	10	16308000	621600	678362	0.0416
s1238	3168	7	189700	35536	46298	0.2441
s1423	1142	6	75750	17920	23758	0.3136
s5378	14644	9	2255470	24032	263665	0.1169
s35932	75473	13	125100800	3413760	5478154	0.0438

6. CONCLUSIONS

In this work, fault dictionary compaction has been addressed from the point of view of two relatively (but not entirely) orthogonal tasks. The first of these tasks has been identified as identifying diagnostically useful information. Previous techniques attempting to solve this problem had several limitations, thus limiting their applicability to large practical circuits. Efficient techniques were proposed to identify diagnostically useful information, thus resulting in dictionaries with satisfactory resolution and are feasible to be generated on large practical circuits.

The second task is one of representing identified information efficiently. This task is especially significant in applications where the first task cannot be performed. This is true in many faulty chips where the fault modeling process is not accurate, i.e., to say that the presence of unmodeled faults could have caused the observed errors. A novel approach to storing all the information in the full fault dictionary based on the use of unlabeled tree encoding has been proposed as an alternative to reducing the size of storage considerably. The proposed storage structure uniquely captures the indistinguishability class information at various stages of the diagnosis process and, hence, is shown to exhibit better behavior than currently known techniques for the purpose of matching the faulty symptoms with the stored data.

However, in spite of the proposed advances, it may be still possible to find applications where it is impossible to maintain any kind of dictionary. This is true in cases when no information may be discarded from the full fault dictionary. In such cases, it may be interesting to investigate the possible use of a combination of multiple stage diagnosis [11] and compact fault dictionaries to provide an effective solution to the diagnosis problem. An important consideration when using fault dictionaries is their applicability in the presence of unmodeled faults. This necessitates the development and use of sophisticated matching algorithms. Currently fault dictionaries store diagnostic output information to guide the fault diagnosis process. It may be fruitful to study the effect of storing information on internal lines of the circuit with the hope of guiding and speeding up the diagnostic reasoning process. Such efforts may lead to new challenges in the compaction of fault dictionaries, but the techniques developed for compacting output information alone may prove to be valuable tools in coping with the problems of the future.

REFERENCES

[1] R. C. Read, *Graph Theory and Computing, NY: Academic Press*, 1972, pp.153-182.

[2] M. Abramovici, M. A. Breuer, and A. D. Friedman, *Digital System Testing and Testable Design, NY: Computer Science Press*, 1990.

[3] H. Y. Chang, E. Manning, and G. Metze, *Fault Diagnosis of Digital Systems, NY: Wiley Interscience*, 1970.

[4] J. Savir and J. P. Roth, "Testing for, and Distinguishing between Failures," *Proc. 12th Intl. Symposium on Fault Tolerant Computing,* Jun. 1982, pp. 165-172.

[5] V. Boppana and W. K. Fuchs, "Fault Dictionary Compaction by the Elimination of Output Sequences," *Proc. IEEE Intl. Conf. on Computer Aided Design*, Nov. 1994, pp. 576-579.

[6] H. Prüfer, "Neuer Beweis eines Satzes über Permutationen," *Arch. Math. Phys.*, Vol. 27, 1918, pp. 742-747.

[7] B. Chess and T. Larrabee, "Test Pattern Generation for Small Full Dictionaries," *Univ. of Santa Cruz, Manuscript,* Apr. 1994.

[8] R. C. Aitken and P. C. Maxwell, "Better Models or Better Alogorithms? Techniques to Improve Fault Diagnosis," *Hewlett-Packard Journal*, Feb. 1995, pp. 110-116.

[9] R. E. Tulloss, "Size Optimization of Fault Dictionaries," *Proc. Semi-conductor Test Conf.*, 1978, pp. 264-265.

[10] J. Richman and K. R. Bowden, "The Modern Fault Dictionary," *Proc. IEEE Intl. Test Conf.*, Sept. 1985, pp. 696-702.

[11] P. G. Ryan, S. Rawat, and W. K. Fuchs, "Two-Stage Fault Location," *Proc. IEEE Intl. Test Conf.*, Oct. 1991, pp. 963-968.

[12] V. Ratford and P. Keating, "Integrated Guided Probe and Fault Dictionary: An Enhanced Diagnostic Approach," *Proc. IEEE Intl. Test Conf.*, 1986, pp. 304-311.

[13] R. E. Tulloss, "Fault Dictionary Compression: Recognizing when a Fault may be Unambiguously Represented by a Single Failure Detection," *Proc. IEEE Intl. Test Conf.*, Nov. 1980, pp. 368-370.

[14] I. Pomeranz and S. M. Reddy, "On the Generation of Small Dictionaries for Fault Location," *Proc. IEEE Intl. Conf. on Computer Aided Design*, Nov. 1992, pp. 272-279.

[15] P. G. Ryan, W. K. Fuchs, and I. Pomeranz, "Fault Dictionary Compression and Equivalence Class Computation for Sequential Circuits," *Proc. IEEE Intl. Conf. on Computer Aided Design*, Nov. 1993, pp. 508-511.

[16] P. G. Ryan, W. K. Fuchs, and I. Pomeranz, "Fault Dictionary Compression," *Manuscript*, 1994.